MIPS Assembly

语法

> *comment: `#`*
>
> *宏:*
>
> ```
> .macro setnum(%des)
>     li $v0 5
>     syscall
>     move %des,$v0
> .end_macro
> .macro printnum(%num)
>     li $v0 1
>     move $a0,%num
>     syscall
> .end_macro
> ```

数据类型

- `.asciiz` ：代表分配一定内存空间存储字符串，字符串最后结尾会加上 `\0` （`.ascii` 不会加上 `\0`），e.g. `myMessage: .asciiz "Hello World!\n"` （字符串表示可 `"" or ''`）
- `.byte` ：一个8bit的值，e.g. `myCharacter: .byte 'm'`
- `.word` ：一个32bit（4byte）的值，e.g. `age: .word 23`
    - `.word value : count` ：初始化预留的空间，并设置 `count` 个 `value` 在新数组中
- `.float` ：一个单精度浮点数，e.g. `PI: .float 3.14`
- `.double` ：一个双精度浮点数，e.g. `myDouble: .double 7.202`
- `.space` ：保留一定bytes的内存空间给指定的变量，可用作字符串或数组，e.g. `userInput: .space 20`

> *`float` 类型在协寄存器中存储4bytes在 Float 一栏*
>
> *`double` 类型在协寄存器中存储8bytes在 Double 一栏，由相邻两个寄存器合并存储，在 Double 栏中合并显示*

指令

- `li` （`load immediate`）：装入立即数常数 `li register_destination, value`
- `la` （`load address`）：加载程序中某些带标点的位置或者变量的地址的宏指令 `la $t0, var1` 即拷贝 `var1` 内存到地址 `t0`
- `lw`（`load word`）：加载 `word`，即该数据大小为4byte

- `lw $1, offset($2)`： `offset` 是一个有符号数，加载的地址是寄存器 `$2$` `+offset` 的值
- `lb`（`load byte`）：加载一个有符号的 8 位字符（同理类比 `lbu`）
- `lwc1`（`load word coprocessor 1(FPU)`）加载到浮点数寄存器而不是整数寄存器
- `ldc1`（`load double coprocessor 1(FPU)`）加载双字节数据存储到寄存器中
- `add.d`：将后两个 `double` 参数相加赋值于第一个参数，e.g. `add.d $f12 $f2 $f0` （同理 `mul.d`）
- `add.s`：将后两个 `float` 参数相加赋值于第一个参数，e.g. `add.s $f12, $f0, $f4`
- `add`：将后两个参数 `word` 参数相加赋值于第一个参数，e.g. `add $t2, $t0, $t1`
- `addi $t0, $t1, 10`：立即数相加
- `move $t0, $t1`：伪指令，其实是 `add $t0, $0, $t1`
- `mul $t1 $t2 $t3`：将 `HI` 设置为高32位， `LO` 和 `$t1` 设置为低32位结果
- `mult $t1, $t2`：将 `HI` 设置为高32位，`LO` 设置为低32位
- `mflo` & `mfhi`：访问 `HI` & `LO` 寄存器
- `sll $t0, $t1, {number of shift}` （`shift left logical`）：左移操作，e.g. `sll $t0, $t1, 2` （同样 `srl`）
- `div $t0, $t1, $t2`：将 `$t1 / $t2` 的结果的整数部分存到 `$t0`
- `div $t0, $t1`：将 `$t0 / $t1` 的结果**商**存到 `lo` **余数**存到 `hi`
- `div.d $f2, $f4, $f6`：将 `$f4 / $f6` 的双精度值存入 `$f2`
- `jal target` （`jump and link`）：跳转至目标地址，并将返回时即将访问的地址存入 `$ra$` 返回地址寄存器中
- `jr $ra`：返回至返回地址寄存器存储的地址位置
- `j label`（`jump`）：无条件跳转
- `sw $t0, offset($t1)`：将 `$t0` 寄存器中的值存储到有效地址 `$t1 + offset` 内存地址中（区别其与 `move` 指令的不同之处）
- `sb`（`store byte`）：同理类比`sw`
- `b`（`branch`）：无条件直接跳转至 `label` 处
- `beq $t0, $t1, label`（`branch equal`）：如果两个寄存器值相等，则跳转 `label` （`bne $t0, $t1, label` 同理）（省略了 `bgt,blt,bltz` 等）
- `slt $a0, $t0, $t1`（`set less than`）：如果 `$t0 < $t1` 则设置 `$a0` 为 `1`，否则设置为 `0` （`sgt $a0, $t0, $t1` 同理）
- `c.eq.s $f0, $f2`：判断两 `float` 数是否相等（同理 `c.eq.d`）
- `bc1t`（`branch coprocessor 1 true`）：判断出来的两个数是否相等 `true/false`

寄存器

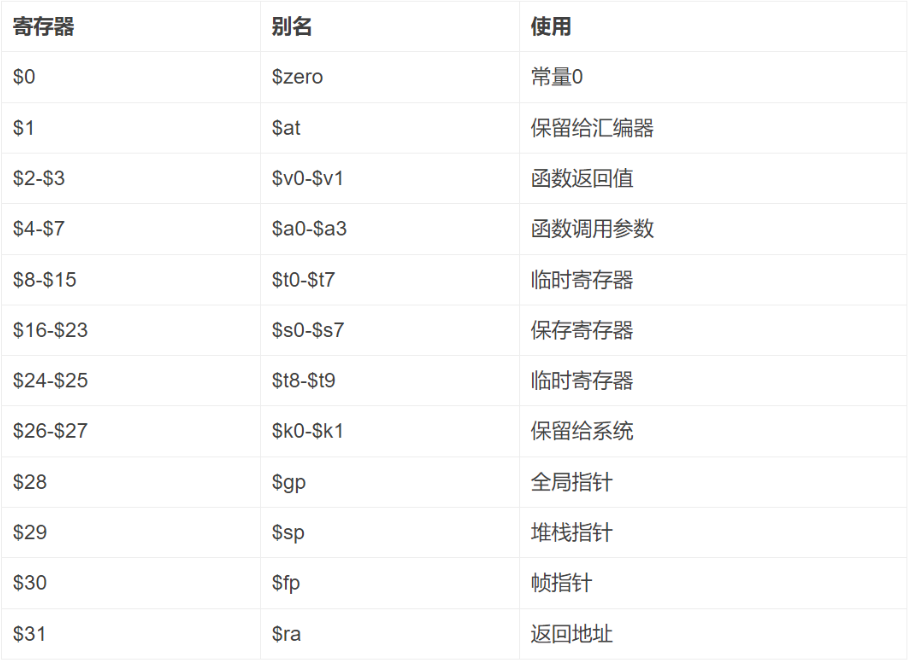
通用寄存器

通用寄存器有32个

- 一般使用 `$v0` 加载服务号

类型	服务号

类型	服务号
[print] Integer	1
[print] Float	2
[print] Double	3
[print] String	4
[read] Integer	5
[read] Float	6
[read] Double	7
[read] String	8
Program Done	10
[print] Character	11
[read] Character	12



寄存器	别名	使用
$0	$zero	常量0
$1	$at	保留给汇编器
$2-$3	$v0-$v1	函数返回值
$4-$7	$a0-$a3	函数调用参数
$8-$15	$t0-$t7	临时寄存器
$16-$23	$s0-$s7	保存寄存器
$24-$25	$t8-$t9	临时寄存器
$26-$27	$k0-$k1	保留给系统
$28	$gp	全局指针
$29	$sp	堆栈指针
$30	$fp	帧指针
$31	$ra	返回地址

P3 Hello Assembly!

代码通常分为两部分

```
.data

.text
```

`.data` 代表变量的声明和分配从这里开始

`.text` 代表程序从这里开始

示例代码

```
.data
    myMessage: .asciiz "Hello, World!\n"

.text
    li $v0, 4
    la $a0, myMessage
    syscall
```

P5 Printing an Integer

注意，对于 `Integer` ，需要输出到 `li $v0, 1`中

```
.data
    age: .word 23

.text
    li $v0, 1
    lw $a0, age
    syscall
```

P6 Printing a Float

注意，对于 `Float` ，需要输出到 `li $v0, 2` ，且需要输出到 `coprocessor 1` 中

```
.data
    PI: .float 3.14

.text
    li $v0, 2
    lwc1 $f12, PI
    syscall
```

P7 Printing a double

注意，使用 `.double` ，**双精度**需要加载 `ldc1`，然后通过 `add.d` 进行输出

```
.data
    myDouble: .double 7.202
    zeroDouble: .double 0.0

.text
    ldc1 $f2, myDouble
    ldc1 $f0, zeroDouble

    li $v0, 3
    add.d $f12, $f2, $f0
    syscall
```

P8 Adding Integers

```
.data
    number1: .word 5
    number2: .word 10

.text
    lw $t0, number1($zero)
    lw $t1, number2($zero)

    add $t2, $t0, $t1

    li $v0, 1
    add $a0, $zero, $t2
    syscall
```

P9 Subtracting Integers

`s0` 和 `s1` 寄存器无法被函数修改

```
.data
    number1: .word 20
    number2: .word 8

.text
    lw $s0, number1
    lw $s1, number2

    sub $t0, $s0, $s1 # t0 = s0 - s1

    li $v0, 1
    move $a0, $t0
    syscall
```

P10 Multiplying Integers `mul`

限制：仅用于两个16位的数相乘，相乘结果为32位，超过此位数不可用 `mul`

```
.data

.text
    addi $s0, $zero, 10
    addi $s1, $zero, 4

    mul $t0, $s0, $s1

    li $v0, 1
    add $a0, $zero, $t0
    syscall
```

P11 Multiplying Integers `mult`

使用 `mult` + `mflo`

```
.data

.text
    addi $t0, $zero, 1000
    addi $t1, $zero, 20

    mult $t0, $t1

    mflo $s0

    # displays the product to the screen
    li $v0, 1
    add $a0, $zero, $s0
    syscall
```

P12 Multiplying Integers `sll`

```
.data


.text
    addi $s0, $zero, 4

    sll $t0, $s0, 2

    li $v0, 1
    add $a0, $zero, $t0
    syscall
```

P13 & P14 Dividing Integers

`div $t0, $t1, $t2`

```
.data

.text
    addi $t0, $zero, 31
    addi $t1, $zero, 5

    div $s0, $t0, $t1

    li $v0, 1
    add $a0, $s0, $zero
    syscall
```

`div $t0, $t1`

```
.data

.text
    addi $t0, $zero, 31
    addi $t1, $zero, 5

    div $t0, $t1

    mflo $s0 # Quotient
    mfhi $s1 # Remainder

    li $v0, 1
    add $a0, $s1, $zero
    # add $a0, $s0, $zero
    syscall
```

P15 Introduction to Functions

```
.data
    message: .asciiz "HI\nhi!!!"

.text
    main:
        jal displayMessage

    # tell the system that the program is done
    li $v0, 10
    syscall

    displayMessage:
        li $v0, 4
        la $a0, message
        syscall

        jr $ra
```

P16 Function Arguments and Return Values

```
.data

.text
    main:
        addi $a1, $zero, 50
        addi $a2, $zero, 100

        jal addNumbers

        li $v0, 1
        addi $a0, $v1, 0
        syscall

    li $v0, 10
    syscall

    addNumbers:
        add $v1, $a1, $a2
        jr $ra
```

P17 Saving Registers to the Stack

- 当使用`$s$`寄存器的时候一定需要将其旧值存入堆栈中，并在函数结束之后将其返回
- 设置`$t and $s`寄存器，就是为了**约定**区分程序员是否可以更改其中的值

一些模板：

- 预留栈空间：

  ```
  addi $sp, $sp, {bytes of all values in $s}

  # e.g. adjust stack to make room for 3 items
  addi $sp, $sp, -12
  ```

- 保存值入堆栈中：

  ```
  # e.g. transport 3 values
  sw $t1, 8($sp)
  sw $t0, 4($sp)
  sw $s0, 0($sp)
  ```

- 旧值回归：

  ```
  lw $s0, 0($sp)
  lw $t0, 4($sp)
  lw $t1, 8($sp)
  addi $sp, $sp, 12
  ```

```
.data
    newLine: .asciiz "\n"

.text
    main:
        addi $s0, $zero, 10

        jal increaseMyRegister

        li $v0, 4
        la $a0, newLine
        syscall

        # print value
```

```
        li $v0, 1
        move $a0, $s0
        syscall

    li $v0, 10
    syscall

    increaseMyRegister:
        addi $sp, $sp, -4
        sw $s0, 0($sp)

        addi $s0, $s0, 30

        li $v0, 1
        move $a0, $s0
        syscall

        lw $s0, 0($sp)
        addi $sp, $sp, 4

        jr $ra
```

P18 Nested Procedures

- 当函数嵌套调用时，`$ra` 的值也需要保存到堆栈中！

```
.data
    newLine: .asciiz "\n"

.text
    main:
        addi $s0, $zero, 10

        jal increaseMyRegister

        li $v0, 4
        la $a0, newLine
        syscall

        # print value
        li $v0, 1
        move $a0, $s0
        syscall

    li $v0, 10
    syscall

    increaseMyRegister:
        addi $sp, $sp, -8
        sw $s0, 0($sp)
        sw $ra, 4($sp)
        addi $s0, $s0, 30

        # Nested Procedure
        jal printTheValue
        lw $ra, 4($sp)

        lw $s0, 0($sp)
        addi $sp, $sp, 8

        jr $ra

    printTheValue:
        #print new value in function
        li $v0, 1
        move $a0, $s0
        syscall

        jr $ra
```

P19 Getting User's Input integer

- 输入Integer，使用服务号**5**
- 输入进来的内容存储在 `$v0` 寄存器中，需要将它移动到其他寄存器！

```
.data
    prompt: .asciiz "Enter your age: "
    message: .asciiz "Your age is: "

.text
    # prompt the user to enter age
    li $v0, 4
    la $a0, prompt
    syscall

    # get user's input
    li $v0, 5
    syscall

    # store the result in $t0
    move $t0, $v0

    # print
    li $v0, 4
    la $a0, message
    syscall

    li $v0, 1
    move $a0, $t0
    syscall
```

P20 Getting User's Input floats

- 输入Float，使用服务号**6**
- 输入进来的内容存储在 `$f0` 寄存器中，需要将它移动到其他寄存器！
- 由于协寄存器无类似 `$zero` 的默认值寄存器，故对于好习惯，将其中有个协寄存器存为 0.0，如 `lwc1 $f4 zeroAsFloat`

```
.data
    message: .asciiz "Enter the value of PI: "
    zeroAsFloat: .float 0.0
.text
    main:
        lwc1 $f4, zeroAsFloat

        # display message
        li $v0, 4
        la $a0, message
        syscall

        # read input
        li $v0, 6
        syscall

        # display the value
        li $v0, 2
        add.s $f12 $f0, $f4
        syscall
```

P21 Getting User's Input doubles

- 输入Double，使用服务号**7**
- 输入进来的内容存储在 `$f0` 寄存器中，需要将它移动到其他寄存器！

```
.data
    message: .asciiz "Enter the value of e: "
    zero: .double 0.0

.text
    main:
        ldc1 $f4, zero

        li $v0, 4
        la $a0, message
        syscall

        # Get the double from the user
        li $v0, 7
        syscall

        # Display user's input
        li $v0, 3
        add.d $f12, $f0, $f4
        syscall
```

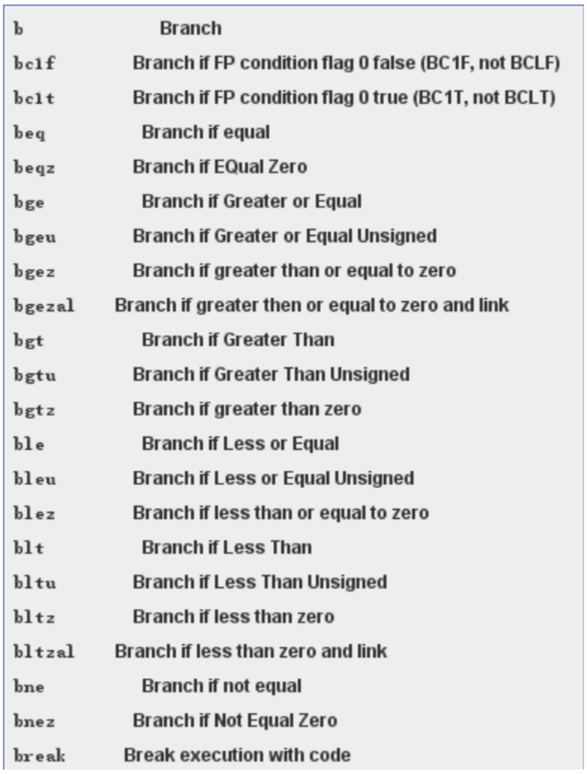
P22 Getting Text From the User

- 输入String，使用服务号**8**
- 将我们的字符串预留数组的地址通过 `la` 传递给 `$a0`
- 并通过 `$a1` 告知数组预留大小
- 通过 `syscall` 进行输入至地址处

```
.data
    message: .asciiz "Hello, "
    userInput: .space 20 # allow user input 20 characters
.text
    main:
        # getting user's input as text
        li $v0, 8
        la $a0, userInput
        li $a1, 20
        syscall

        # display hello
        li $v0, 4
        la $a0, message
        syscall

        # display the name
        li $v0, 4
        la $a0, userInput
        syscall

    li $v0, 10
    syscall
```

P23 If statements Branching Instructions

- 对于 **if - else** 语句，我们需要两个 `label` ，第一个用于**条件满足**，第二个用于条件不满足，然后执行事务后跳转至指定位置（或者其他跳转也是可以的，不然第一个 `label` 跳转之后就会**继续往后执行**）



b	Branch
bc1f	Branch if FP condition flag 0 false (BC1F, not BCLF)
bc1t	Branch if FP condition flag 0 true (BC1T, not BCLT)
beq	Branch if equal
beqz	Branch if EQual Zero
bge	Branch if Greater or Equal
bgeu	Branch if Greater or Equal Unsigned
bgez	Branch if greater than or equal to zero
bgezal	Branch if greater then or equal to zero and link
bgt	Branch if Greater Than
bgtu	Branch if Greater Than Unsigned
bgtz	Branch if greater than zero
ble	Branch if Less or Equal
bleu	Branch if Less or Equal Unsigned
blez	Branch if less than or equal to zero
blt	Branch if Less Than
bltu	Branch if Less Than Unsigned
bltz	Branch if less than zero
bltzal	Branch if less than zero and link
bne	Branch if not equal
bnez	Branch if Not Equal Zero
break	Break execution with code

```
.data
    message: .asciiz "The numbers are equal."
    message1: .asciiz "The numbers are different."
    message2: .asciiz "Nothing happened."
    finish: .asciiz "Finished!"

.text
    main:
        li $t0, 20
        li $t1, 20
        # addi $t0, $zero, 5
        # addi $t1, $zero, 20

        beq $t0, $t1, numbersEqual
        li $v0, 4
        la $a0, message1
        syscall
        b Nxt

    numbersEqual:
        li $v0, 4
        la $a0, message
        syscall

    Nxt:
        li $v0, 4
        la $a0, finish
        syscall

    li $v0, 10
    syscall
```

```
# same result
.data
    message: .asciiz "The numbers are equal."
    message1: .asciiz "The numbers are different."
    message2: .asciiz "Nothing happened."
    finish: .asciiz "Finished!"

.text
    main:
        li $t0, 20
        li $t1, 20
        # addi $t0, $zero, 5
        # addi $t1, $zero, 20

        beq $t0, $t1, numbersEqual
        li $v0, 4
        la $a0, message1
        syscall

    next:
        li $v0, 4
        la $a0, finish
        syscall

    li $v0, 10
    syscall

    numbersEqual:
        li $v0, 4
        la $a0, message
        syscall

        b next
```

P24 Checking If a Number is Less than Another `slt`

```
.data
    messageLess: .asciiz "The number is less than the other."
    messageGreater: .asciiz "The number is greater than the other."

.text
    main:
        li $t0, 300
        li $t1, 200

        slt $s0, $t0, $t1
        bne $s0, $zero, printMessageLess
        jal printMessageGreater
        next:

    li $v0, 10
    syscall

    printMessageLess:
        li $v0, 4
        la $a0, messageLess
        syscall

        b next

    printMessageGreater:
        li $v0, 4
        la $a0, messageGreater
        syscall

        jr $ra
```

P26 While Loop in MIPS

- 使用 `whlie label & exit label` 实现 **While Loop**

```
.data
    message: .asciiz "After while loop is done."
    newLine: .asciiz "\n"

.text
    main:
        # i = zero
        addi $t0, $zero, 0

        while:
            bgt $t0, 9, exit

            jal printInteger

            addi $t0, $t0, 1 # i++

            j while

        exit:
            li $v0, 4
            la $a0, message
            syscall


    li $v0, 10
    syscall

    printInteger:
        li $v0, 1
        add $a0, $zero, $t0
        syscall

        li $v0, 4
        la $a0, newLine
        syscall

        jr $ra
```

P27 Arrays

- 使用 `.space` 类型预留内存空间给数字
- 之后通过移动一个 `offset` 来实现逐个存储

```
.data
    myArray: .space 12 # for 3 integers

.text
    main:
        addi $s0, $zero, 4
        addi $s1, $zero, 10
        addi $s2, $zero, 12

        # Index = $t0
        addi $t0, $zero, 0

        sw $s0, myArray($t0)
            addi $t0, $t0, 4
        sw $s1, myArray($t0)
            addi $t0, $t0, 4
        sw $s2, myArray($t0)

        lw $t6, myArray($zero)

        li $v0, 1
        addi $a0, $t6, 0
        syscall
```

P28 Printing an Array with a While Loop

```
.data
    myArray: .space 12 # for 3 integers
    newLine: .asciiz "\n"

.text
    main:
        addi $s0, $zero, 4
        addi $s1, $zero, 10
        addi $s2, $zero, 12
        # Index = $t0
        addi $t0, $zero, 0

        sw $s0, myArray($t0)
            addi $t0, $t0, 4
        sw $s1, myArray($t0)
            addi $t0, $t0, 4
        sw $s2, myArray($t0)

        lw $t6, myArray($zero)

        # while loop
        addi $t0, $zero, 0 # clear $t0

        while:
            beq $t0, 12, exit

            lw $t6, myArray($t0)

            li $v0, 1
            addi $a0, $t6, 0
            syscall

            li $v0, 4
            la $a0, newLine
            syscall

            addi $t0, $t0, 4

            j while

        exit:

    # tell system is end of program
    li $v0, 10
    syscall
```

P29 Array Initializer

- 使用 `.word value:count` 来初始化数组

```
.data
    myArray: .word 100:3
    newLine: .asciiz "\n"

.text
    main:
        # while loop
        addi $t0, $zero, 0 # clear $t0

        while:
            beq $t0, 12, exit

            lw $t6, myArray($t0)

            li $v0, 1
            addi $a0, $t6, 0
            syscall

            li $v0, 4
            la $a0, newLine
            syscall

            addi $t0, $t0, 4

            j while

        exit:

    # tell system is end of program
    li $v0, 10
    syscall
```

P30 Floating Point Arithmetic

- 当处理 `double` 浮点数时，不要存在**相邻的两个协寄存器**中
- 因为 `double` 会占用相邻的两个寄存器存取单个值

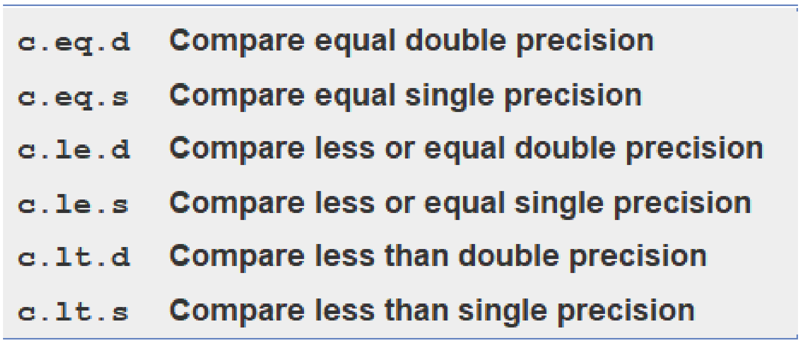
```
.data
    number1: .double 3.14
    number2: .double 2.71

.text
    main:
        ldc1 $f2, number1
        ldc1 $f4, number2

        li $v0, 3
        add.d $f12, $f2, $f4
        syscall
```

P31 More about Floating Point Arithmetic

```
.data
    number1: .double 3.14
    number2: .double 2.71

.text
    main:
        ldc1 $f2, number1
        ldc1 $f4, number2

        li $v0, 3
        div.d $f12, $f2, $f4
        syscall
```

P32 If Statement with Floats and Doubles

- If statement for precision



c.eq.d	Compare equal double precision
c.eq.s	Compare equal single precision
c.le.d	Compare less or equal double precision
c.le.s	Compare less or equal single precision
c.lt.d	Compare less than double precision
c.lt.s	Compare less than single precision

```
.data
    message: .asciiz "It was true.\n"
    message2: .asciiz "It was false.\n"
    number1: .float 10.4
    number2: .float 10.4

.text
    main:
        lwc1 $f0, number1
        lwc1 $f2, number2

        c.eq.s $f0, $f2

        bc1t exit

        li $v0, 4
        la $a0, message2
        syscall

    li $v0, 10
    syscall

    exit:
        li $v0, 4
        la $a0, message
        syscall
```

P33&34 Introduction to Recursion

（oh，MIPS的递归参数传值真是难搞啊😅）

```
.data
    prompMessage: .asciiz "Enter a number to find its factorial: "
    resultMessage: .asciiz "\nThe factorial of the number is "
    number: .word 0
    answer: .word 0

.text
    .globl main
    main:
        # read the number from the user
        li $v0, 4
        la $a0, prompMessage
        syscall

        li $v0, 5
        syscall

        sw $v0, number

        # call the factorial function
        lw $a0, number
        jal findFactorial
        sw $v0, answer

        # display the results
        li $v0, 4
        la $a0, resultMessage
        syscall

        li $v0, 1
        lw $a0, answer
        syscall

    li $v0, 10
    syscall

#----------------------------------------

    .globl findFactorial
    findFactorial:
        # save the args of last recursion
        subu $sp, $sp, 8
        sw $ra, ($sp)
        sw $s0, 4($sp)

        # move the current args to s registers
        move $s0, $a0

        # base case
        li $v0, 1
        beq $a0, 0, factorialDone

        # call function
        sub $a0, $a0, 1
        jal findFactorial

        # renew the $v0 register
        mul $v0, $a0, $v0

        # jump registers
        factorialDone:
            lw $ra, ($sp)
            lw $a0, 4($sp)
            addu $sp, $sp, 8
            jr $ra
```

P35 Bit Manipulation

- 使用左右移操作制作 mask 来进行位操作

```
.data
    newLine: .asciiz "\n"

.text
    main:
        li $a1, 11
        jal showNumber

        li $a1, 11
        jal clearBitZero

        move $a1, $v0
        jal showNumber

    li $v0, 10
    syscall

    showNumber:
        li $v0, 4
        la $a0, newLine
        syscall

        li $v0, 1
        move $a0, $a1
        syscall

        jr $ra

    clearBitZero:
        addi $sp, $sp, -4
        sw $s0, 0($sp)

        li $s0, -1
        sll $s0, $s0, 1
        and $v0, $a1, $s0

        lw $s0, 0($sp)
        addi $sp, $sp, 4

        jr $ra
```

P36 Average Program

```
.data
    array: .word 10, 2, 9
    length: .word 3
    sum: .word 0
    average: .word 0

.text
    main:
        la $t0, array # base address
        li $t1, 0 # i = 0
        lw $t2, length # t2 = length
        li $t3, 0 # sum = 0

        sumLoop:
            lw $t4, ($t0) # t4 = array[i]
            add $t3, $t3, $t4 # sum = sum + array[i]

            add $t1, $t1, 1 # i = i + 1
            add $t0, $t0, 4 # updating the array address
            blt $t1, $t2, sumLoop # if i < len, then loop again

        sw $t3, sum

        div $t5, $t3, $t2
        sw $t5, average

        li $v0, 1
        lw $a0, average
        syscall

    li $v0, 10
    syscall
```

example

`leapJudge`

```
# if(x % 400 == 0) GOTO_YES
# else if(x % 100 == 0) GOTO_NO
# else if(x % 4 == 0) GOTO_YES
.data

.text
    main:
        li $t1, 400
        li $t2, 100
        li $t3, 4

        li $v0, 5
        syscall
        move $s0, $v0

        div $s0, $t1
        mfhi $t0
        beq $t0, 0, GOTO_YES
        div $s0, $t2
        mfhi $t0
        beq $t0, 0, GOTO_NO
        div $s0, $t3
        mfhi $t0
        beq $t0, 0, GOTO_YES
        b GOTO_NO

    EXIT:
    li $v0, 10
    syscall

    GOTO_YES:
        li $v0, 1
        li $a0, 1
        syscall

        j EXIT

    GOTO_NO:
        li $v0, 1
        li $a0, 0
        syscall

        j EXIT
```

`primeJudge`

```
.data

.text
    main:
        li $v0, 5
        syscall
        move $s0, $v0   # $s0: m

        beq $s0, 1, GOTO_NO

        li $t0, 2
        FOR_1:
            beq $t0, $s0, END_FOR_1

            div $s0, $t0
            mfhi $t1
            beq $t1, 0, GOTO_NO

            addi $t0, $t0, 1
            j FOR_1

    END_FOR_1:
        j GOTO_YES


    EXIT:
    li $v0, 10
    syscall

    GOTO_YES:
        li $v0, 1
        li $a0, 1
        syscall

        j EXIT

    GOTO_NO:
        li $v0, 1
        li $a0, 0
        syscall

        j EXIT
```

`palindromeJudge`

```
.data
    str: .space 100

```

```
.text
    main:
            li $v0, 5
            syscall
            move $s0, $v0   # $s0: length of string

            li $t0, 0
            FOR_1:
                beq $t0, $s0, END_FOR_1

                li $v0, 12
                syscall
                sb $v0, str($t0)

                addi $t0, $t0, 1

            j FOR_1
        END_FOR_1:

            li $t0, 0
            FOR_2:
                beq $t0, $s0, GOTO_YES

                sub $t1, $s0, $t0
                subi $t1, $t1, 1

                lb $t2, str($t0)
                lb $t3, str($t1)

                bne $t2, $t3, GOTO_NO

                addi $t0, $t0, 1
                j FOR_2

            GOTO_YES:
                li $v0, 1
                li $a0, 1
                syscall
                j EXIT

            GOTO_NO:
                li $v0, 1
                li $a0, 0
                syscall
                j EXIT

            EXIT:

    li $v0, 10
    syscall
```

`stringPartialReverse`

```
.data
    arr: .space 1024

.text
    main:
        li $v0, 5
        syscall
        move $s0, $v0   # $s0 = length

        li $v0, 5
        syscall
        move $s1, $v0   # $s1 = start pointer

        li $v0, 5
        syscall
        move $s2, $v0   # $s2 = end pointer

        li $v0, 8
        la $a0, arr # address of string
        li $a1, 1024    # reserve bytes
        syscall

        while:
            bgt $s1, $s2, exit
            lb $t1, arr($s1)
            lb $t2, arr($s2)
            sb $t1, arr($s2)
            sb $t2, arr($s1)

            addi $s1, $s1, 1
            addi $s2, $s2, -1
            j while
    exit:
        li $v0, 4
        la $a0, arr
        syscall

    li $v0, 10
    syscall
```

`Catalan`

```
# int catalanArray[20] = {1, 1};
# int n;
# scanf("%d", &n);
# for(int i = 2; i <= n; i++) {
#   for(int j = 0; j < i; j++) {
```

```
#       catalanArray[i] += catalanArray[j] * 
catalanArray[i - j - 1];
#       printf("%d\n", catalanArray[i]);
#   }
# }
# printf("catalan[%d]=%d\n", n, catalanArray[n]);

.data
    array: .space 1000
    outputPrompt_1: .asciiz "catalan["
    outputPrompt_2: .asciiz "]="
    newLine: .asciiz "\n"

.text
    main:
        # set initial value of location 0/1
        li $t0, 1
        li $t1, 0
        sw $t0, array($t1)
        addi $t1, $t1, 4
        sw $t0, array($t1)

        # start process
        li $v0, 5
        syscall
        move $s0, $v0   # $s0: n

        li $t0, 2   # $t0: i
        FOR_i:
            bgt $t0, $s0, END_FOR_i

            sll $t2, $t0, 2

            li $t1, 0   # $t1: j
            FOR_j:
                beq $t1, $t0, END_FOR_j

                sll $t3, $t1, 2 # j
                sub $t4, $t2, $t3
                subi $t4, $t4, 4    # i - j - 1

                lw $t3, array($t3)
                lw $t4, array($t4)
                mult $t3, $t4
                mflo $t3
                lw $t5, array($t2)
                add $t5, $t5, $t3
                sw $t5, array($t2)

                # print partly
                li $v0, 1
                lw $a0, array($t2)
                syscall
                li $v0, 4
                la $a0, newLine
                syscall

                addi $t1, $t1, 1
                j FOR_j
        END_FOR_j:

            addi $t0, $t0, 1
            j FOR_i
    END_FOR_i:

    li $v0, 4
    la $a0, outputPrompt_1
    syscall
    li $v0, 1
    move $a0, $s0
    syscall
    li $v0, 4
    la $a0, outputPrompt_2
    syscall
    sll $t0, $s0, 2
    lw $a0, array($t0)
    li $v0, 1
    syscall

    li $v0, 10
    syscall
```

gcd

```
.data
    promptShow: .asciiz "Enter two numbers: \n"
    resultShow: .asciiz "The gcd of two numbers is: "

.text
    main:
        # li $v0, 4
        # la $a0, promptShow
        # syscall

        li $v0, 5 # integer 1
        syscall
        move $t0, $v0

        li $v0, 5 # integer 2
        syscall
        move $t1, $v0
        # return (!b) ? a : gcd(b, a % b);
        # while(b != 0) {
        #   int temp = a;
        #   a = b;
        #   b = temp % b;
        # }

        while:
            beq $t1, $zero, exit

            move $t2, $t0
            move $t0, $t1
            div $t2, $t1
            mfhi $t1

            j while
        exit:

        # li $v0, 4
        # la $a0, resultShow
        # syscall

        li $v0, 1
        move $a0, $t0
        syscall

    li $v0, 10
    syscall
```

permutation

```
.data
    array: .space 100
    visited: .space 100
    constant_1: .word 1
    newLine: .asciiz "\n"
    newSpace: .asciiz " "

.text
    main:
        li $v0, 5
        syscall
        move $s0, $v0   # $s0: n

        jal Permutation

    li $v0, 10
    syscall

    Permutation:
```

```
    # process
    bgt $s0, $a2, GOTO_1    # $a2: dep
    PRINT_ANSWER:
    li $t0, 0
    FOR_1:
        beq $t0, $s0, END_FOR_1

        sll $t1, $t0, 2
        li $v0, 1
        lw $a0, array($t1)
        syscall

        li $v0, 4
        la $a0, newSpace
        syscall

        addi $t0, $t0, 1
        j FOR_1
END_FOR_1:
    li $v0, 4
    la $a0, newLine
    syscall

    jr $ra

    GOTO_1:
    li $t0, 0
    FOR_2:
        beq $t0, $s0, END_FOR_2

        sll $t1, $t0, 2 # address i
        addi $t5, $t0, 1    # integer i + 1
        sll $t2, $t5, 2 # address i + 1
        sll $t3, $a2, 2 # address dep

        lw $t4, visited($t1)    # integer visited[i]
        bne $t4, $zero, GOTO_2

        sw $t5, array($t3)
        lw $t6, constant_1
        sw $t6, visited($t1)

        subi $sp, $sp, 4
        sw $ra, 0($sp)
        subi $sp, $sp, 4
        sw $a2, 0($sp)
        subi $sp, $sp, 4
        sw $t0, 0($sp)
        subi $sp, $sp, 4
        sw $t1, 0($sp)
```

```
            addi $a2, $a2, 1

            jal Permutation

            lw $t1, 0($sp)
            addi $sp, $sp, 4
            lw $t0, 0($sp)
            addi $sp, $sp, 4
            lw $a2, 0($sp)
            addi $sp, $sp, 4
            lw $ra, 0($sp)
            addi $sp, $sp, 4

            sw $zero, visited($t1)

            GOTO_2:
            addi $t0, $t0, 1
            j FOR_2
    END_FOR_2:

        # return with void
        jr $ra
```

primeCount

```
# int prime[100];
# int cnt = 0;
# int start = 2, end = 100;
#
# for(int i = start; i < end;  i++) {
#   for(int j = 2; j < i; j++) {
#       if(i % j == 0) break; // j outer_loop
#   }
#   prime[cnt] = i;
#   cnt++;
# }
.data
    primeArray: .space 40
    inputPrompt: .asciiz "please input the range: \n"
    newLine: .asciiz "\n"

.text
    # [start, end)
    # $t0, start
    # $t1, end
    # $t2, addressStep
    # $t5, count
    # $t3, divisor
    # $t4, remainder
main:
        li $v0, 4
        la $a0, inputPrompt
        syscall

        li $v0, 5
        syscall
        move $t0, $v0

        li $v0, 5
        syscall
        move $t1, $v0

        li $t2, 0
        li $t5, 0

        Loop_1:

            beq $t0, $t1, Done_1

            li $t3, 2

            Loop_2:
                beq $t3, $t0, addPrime

                div $t0, $t3
                mfhi $t4
                beqz $t4, Done_2

                addi $t3, $t3, 1
                j Loop_2

        addPrime:

            sw $t0, primeArray($t2)
            addi $t2, $t2, 4
            addi $t5, $t5, 1

        Done_2:
             addi $t0, $t0, 1
             j Loop_1

    Done_1:
        li $t6, 0
        outputLoop:
            beq $t6, $t2, doneOutput

            li $v0, 1
            lw $a0, primeArray($t6)
            syscall

            li $v0, 4
            la $a0, newLine
```

```
            syscall

            addi $t6, $t6, 4

            j outputLoop

    doneOutput:


li $v0, 10
syscall
```